Chapter 2

Variant's assessment

RQ2. To what extent the generated variants are different ?

In this chapter, we investigate whether the artifically created variants are different. We propose a methodology to compare the program variants both statically and during runtime. Besides, we present a novel study on code preservation, demonstrating that the code transformations introduced by CROW are resilient to later compiling transformations during machine code generation. We evaluate the variant's assessment in both existing scenarios for WebAssembly, browsers and standalone engines.

2.1 Later compiler transformations

WebAssembly is an intermediate language, this means that compilers used it to produce machine code. For program variants, this means that compilers can undo artificial introduced transformations, for example, through optimization passes. When a code transformation is maintained from the first time it is introduced to the final machine code generation is a preserved variant.

Part of the contributions of this thesis are our strategies to prevent reversion of code transformations. We take engineering decision regarding this in all the stages of the CROW workflow. First, we disable all optimizations inside CROW in the generation of the WebAssembly binaries. This prevents the LLVM toolchain used to remove some introduced transformations. However, the LLVM toolchain applies optimizations by default, such as constant folding or logical operations' normalization. As we illustrate previously, these are some transformations found and applied by CROW. We modified the LLVM backend for WebAssembly to avoid this reversion during the creation of Wasm binaries.

TODO Add link to repository here

This phenomenon is sometimes bypassed by diversification studies when they are conducted at high-level. As another contribution, we conduct a study on preservation for both scenarios where Wasm is used, browsers and standalone engines. In

Name	Properties
V8 []	V8 is the engine used by Chrome and NodeJS to execute JavaScript and WebAssembly. TODO Explain compilation process
wasmtime []	Wasmtime is a standalone runtime for WebAssembly. This engine is used by the Fastly platform to provide Edge-Cloud computing services. TODO Explain compilation process

Table 2.1: Wasm engines used during the diversification assessment study. The table is composed by the name of the engine and the description of the compilation process for them.

2.2 Metrics

2.2.1 Static

2.2.2 Dynamic

2.2.2.1 Trace based

2.2.2.2 Timing

2.2.3 Preservation

2.2.3.1 Browser based study (V8)

2.2.3.2 Backend based study (wasmtime)

2.3 Evaluation

2.3.1 Corpora

To answer RQ2, we use the corpora proposed in chapter 1.

2.3.2 Setup

2.4 Results

2.5 Conclusions

Appended papers